In Proc. 19th Digital Avionics Systems Conference, 7-13 October 2000, Philadelphia, PA.

DEVELOPING HIGH ASSURANCE AVIONICS SYSTEMS WITH THE SCR REQUIREMENTS METHOD*

R. Bharadwaj and C. Heitmeyer, Naval Research Laboratory, Wash., DC 20375

Introduction

In high assurance avionics systems, such as systems for flight guidance, air traffic control, and collision avoidance, compelling evidence is required that the system behavior satisfies certain critical properties. Some critical properties are *functional* properties, properties of the services that the system delivers. For example, when another aircraft flies too close, a collision avoidance system must advise the pilot to move the aircraft up or down to avoid a collision. Besides functional properties, four other classes of critical system properties may be identified: *security*, *safety*, *real-time*, and *fault-tolerance*. In most cases, an avionics system must satisfy properties in more than a single class. For example, a collision avoidance system must satisfy not only functional properties, but real-time constraints, fault-tolerance properties, and safety properties.

Researchers have proposed numerous approaches for specifying, constructing, and certifying high assurance systems. These include formal specification notations, formal models, and rigorous verification and validation techniques. But, two difficult problems remain. The first is the need for technology to support the application of these new methods to practical systems. Without such technology, opportunities to transfer basic research results to practice are severely limited. Also needed is a unified framework for building systems that satisfy multiple critical properties. This need exists because not one but several different approaches for developing high assurance systems have evolved, each with a different philosophy of system development and different techniques for specification and assurance.

This paper describes a framework, based on the SCR (Software Cost Reduction) requirements method, that has recently been developed for building high assurance systems. The SCR framework includes a formal specification notation, a state-based formal model, and assurance methods useful for constructing systems that must provide critical services in a secure, safe, timely, and fault-tolerant manner. To illustrate the application of SCR to avionics systems, this paper presents an SCR requirements specification of a small avionics system, introduced by Miller in [1], which powers on a device of interest when the altitude of an aircraft falls below a specified threshold. To develop a specification for this system, we follow a four-step process. This process provides a systematic approach to developing and organizing a requirements specification of a nontrivial system.

*This work is supported by the Office of Naval Research.

The SCR Requirements Method

The SCR (Software Cost Reduction) requirements method is a formal method based on tables for the specification and analysis of the required behavior of safety-critical software systems. Originally developed in 1978 by NRL (Naval Research Laboratory) to document the flight program requirements of the Navy's A-7 aircraft, SCR has also been applied by a number of organizations in industry (e.g., Grummann, Ontario Hydro, Bell Laboratories, and Lockheed) to a wide range of practical systems, including avionics and space systems. For example, in 1994, in the largest application of SCR to date, Lockheed used SCR to specify the requirements of the C-130J flight control software, which contains more than 250,000 lines of Ada code.

To provide tool support for the SCR method, our group at NRL has developed an integrated suite of tools called the SCR* toolset [2]. The toolset includes a *consistency checker* for checking the specification for type errors, missing cases, and

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2000**	2. REPORT TYPE	3. DATES COVERED **00-00-2000 to 00-00-2000**
4. TITLE AND SUBTITLE **Developing High Assurance Avionics Systems with the SCR Requirements Method**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,4555 Overlook Avenue, SW,Washington,DC,20375**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**8**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

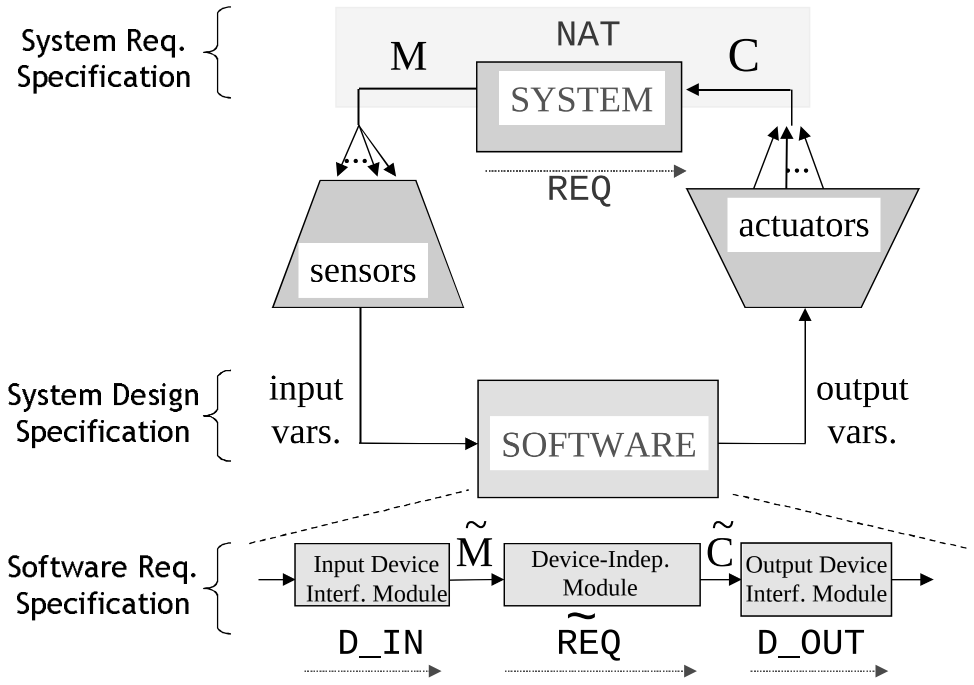
System Req. Specification
System Design Specification
Software Req. Specification
NAT
M
C
SYSTEM
REQ
sensors
actuators
input vars.
output vars.
SOFTWARE
$\tilde{M}$
$\tilde{C}$
Input Device Interf. Module
Device-Indep. Module
Output Device Interf. Module
D_IN
$\widetilde{REQ}$
D_OUT

Figure 1. Relationship between the SRS, the SDS, and the SoRS

unwanted nondeterminism [3]; a *dependency graph browser* for displaying the dependencies among the variables in the specification; and a *simulator* for symbolically executing the system based on the specification. Currently, more than 100 academic, industrial, and government organizations in the US, Canada, and several other countries are experimenting with the SCR* toolset.

The utility of the SCR* toolset has been evaluated in a number of pilot projects. In one, NASA researchers used the toolset's consistency checker to detect several errors in the requirements specification of software for the International Space Station [4]. In a second project, engineers at Rockwell used the tools to expose 28 errors, many of them serious, in the requirements specification of a flight guidance system [5]. Of the detected errors, a third were uncovered by creating the specification with the toolset, a third in running the consistency checker, and the remaining third in executing the specification with the simulator.

Process for Specifying Requirements

Figure 1 is the basis for a four-step process for constructing a requirements specification. The first step creates the System Requirements Specification (SRS), which describes the required external behavior of the system in terms of monitored and controlled quantities in the system environment. We refer to the behavior described by the SRS as the "ideal" system behavior because it omits any mention of I/O devices and hardware failures. The remaining steps refine and extend the SRS. Step 2 creates the System Design Specification (SDS), which identifies the system's input and output devices (e.g., sensors and actuators). Step 3 creates the Software Requirements Specification (SoRS), which refines the SRS by adding modules which use values read from input devices to calculate values of the monitored quantities and which use the computed values of controlled quantities to drive output devices. Step 4 extends the SRS by adding behavior that reports hardware malfunctions, e.g., sensor failures.

In this process, we apply the information hiding principle to the requirements specification so that parts of the specification that are unlikely to change together are assigned to different modules. In applying information hiding to the specification, all of the ways in which the requirements are likely to change are identified, and the required system behavior is decomposed into modules so that exactly one module is associated with a single change. The goal is to organize the requirements specification so that each change requires a change to only a single module. How this is achieved is described below.

System Requirements Specification

To construct an SRS with SCR, environmental quantities relevant to the system are identified, and each quantity is represented by a mathematical variable. These quantities consist of both *controlled quantities*, environmental quantities that the system controls, and *monitored quantities*, environmental quantities that can influence system behavior. In Figure 1, M represents the monitored quantities and C represents the controlled quantities.

The desired *system behavior* is documented in the SRS by describing two relations, NAT and REQ, on the monitored and controlled quantities; these relations are part of the Parnas Four Variable Model [6]. The relation NAT describes the constraints imposed on the environmental quantities by physical laws and the system environment. REQ describes the relation between the monitored and controlled quantities that the system must enforce to produce the required behavior. In developing the SRS, we initially specify REQ in terms of the ideal behavior of the system; that is, we assume that the system can obtain perfect values of the monitored quantities and compute perfect values of the controlled quantities. Later, for each controlled variable, we specify timing constraints (and possibly tolerances).

System Design Specification

The SDS identifies and documents the characteristics of the resources that are available to estimate values of the monitored quantities and to set values of the controlled quantities. These values are usually read from or written to hardware devices, such as sensors and actuators. The values in the system's hardware/software interfaces are denoted by variables. These variables are partitioned into *input variables,* values read by input devices, and *output variables*, values written to output devices. The product of this step is a description of the input and output devices and of the relationship between the input and output variables and the monitored and controlled variables.

Software Requirements Specification

The SRS and the SDS are the foundation for the SoRS, which describes how the input variables are to be used to estimate values of the monitored variables, and how estimates of the controlled variables are to be used to control the output devices using the output variables. Figure 1 shows the relationship between the SRS, the SDS, and the SoRS. It also shows the decomposition of the SoRS into three modules: two *device-dependent* modules called the *input device interface module* and the *output device interface module*, and a single device-independent module called *the function driver module*. This organization was influenced by the module structure of the A-7 flight software. In Figure 1, relation D_IN specifies how estimates of the monitored variables, represented by $\tilde{M}$, are computed in terms of the input variables, and relation D_OUT specifies how estimates of the controlled variables, represented by $\tilde{C}$, are used to compute the values of the output variables. The outputs of the input device interface module, i.e., the estimated values of the monitored variables, form the inputs to the function driver module. The function driver module uses these estimates to compute estimates of the controlled variables.

The required behavior of the function driver module is already defined by the REQ relation, specified as part of the SRS during Step 1 of our process. What remains is to document the required behavior of the device-dependent modules, i.e., D_IN and D_OUT. To satisfy the information hiding principle, the input device interface module only uses values of input variables to estimate values of the monitored variables, and the output device interface module only uses values of controlled variables to compute the values of the output variables. The benefit of this approach is that it makes the specification easy to change. For example, to replace an input or output device with a new device or to modify or add a system function, usually only a small part of a single module will change.

In Figure 1, the relation $\tilde{REQ}$ specifies the relation between estimates of the monitored quantities $\tilde{M}$ and estimates of the controlled quantities $\tilde{C}$. In most cases, $\tilde{REQ}$ extends REQ because $\tilde{REQ}$ not only describes the ideal behavior captured by REQ but also describes the externally visible behavior that is not part of the ideal behavior. Because REQ is based on perfect knowledge of the monitored quantities and perfect computations of the controlled quantities, REQ does not describe how the system reports hardware malfunctions. In practical systems, hardware devices, such as sensors, will fail, and the system will need to provide external notification of such failures. $\tilde{REQ}$ extends the required behavior described by REQ by describing how notification of hardware malfunctions is presented to system users.

The SCR Notation

To specify the required system behavior in a practical and efficient manner, the SCR method uses mode classes. A *mode class*, whose values are modes, is an auxiliary variable that helps keep the specification concise. Each mode defines an equivalence class of system states useful in specifying the required system behavior. In SCR specifications, we often use the following prefixes in variable names: "m" to indicate monitored variables, "mc" for mode classes, "c" for controlled variables, "i" for input variables, and "o" for output variables.

Conditions and events are important constructs in SCR specifications. A *condition* is a predicate defined on one or more state variables (a *state variable* is a monitored or controlled variable or a mode class). An *event* occurs when a state variable changes value. A *conditioned event* has the form "@T(c) WHEN d" and is defined by the expression "NOT c AND c' and d," where the unprimed conditions c and d are evaluated in the "old" state, and the primed condition c' is evaluated in the "new" state. Informally, this expression denotes the event "predicate c becomes true in the new state when predicate d holds in the old state." The notation "@F(c)" denotes the event @T(NOT c). The notation DUR(c) indicates the length of time that condition c has been continuously true.

To specify the REQ, D_IN, and D_OUT relations, SCR specifications use a set of tables. Each table defines the value of a dependent variable (a mode class or controlled variable) as a function. A table may be either a condition table or an event table. Typically, a condition table describes the value of a controlled variable as a function of a mode class and a *condition*, whereas an event table describes the value of a controlled variable as a function of a mode class and an *event*. A mode transition table is a special case of an event table. Although many SCR tables use modes to define the value of a variable, some SCR tables omit modes.

Specifying the ASW in SCR

To illustrate the above process, we apply it to the Altitude Switch (ASW) described in [1]. The ASW turns on the power to a Device of Interest (DOI) when the aircraft descends below a threshold altitude of 2000 feet. The ASW accepts an inhibit signal that prevents it from turning on power to the DOI, and a reset signal that returns the system to its initial state. It receives altitude information from an analog altimeter and two digital altimeters. If the DOI fails to turn on within two seconds, if all three altimeters fail for more than two seconds, or if system initialization fails, the ASW indicates a fault by turning on a fault indicator light and by failing to strobe a watchdog timer.

This section demonstrates how a requirements specification for the ASW can be developed using the four-step process described above. First, the ideal behavior of the ASW is described by specifying the monitored and controlled variables and the relation REQ. The product of Step 1 is the SRS. Specified in Step 2 are the input and output variables associated with the ASW I/O devices and the relationship between these variables and the monitored and controlled variables. The third step describes the SoRS by specifying the relations D_IN and D_OUT, i.e., how estimates of the monitored variables are computed from values of the input variables and how the values of the controlled variables are written to output devices.

Finally, the SoRS and the SRS are extended to support the reporting of hardware malfunctions.

ASW System Requirements Specification

To specify the ideal behavior of the ASW, the SRS defines seven variables: five monitored variables, one controlled variable, and one mode class. Described below are four steps we follow to specify the ideal behavior: 1) describe the controlled variables; 2) describe the monitored variables; 3) describe the mode classes, and 4) describe the required relation REQ between the monitored and controlled variables.

In the ASW, the single controlled quantity is the signal that wakes up the DOI. This is represented by the controlled variable cWakeupDOI which has an initial value of false. Table 1 lists the single ASW controlled variable, its type, initial value, and a brief description.

Table 1. Controlled Variables of the ASW

Name	Type	Init. Value	Description
cWakeupDOI	boolean	false	True means power on DOI

Described next are the environmental quantities that the ASW monitors to determine when to turn the power to the DOI on. Table 2 lists five monitored variables along with their types, initial values, and brief descriptions. The monitored variable mAltBelow represents the aircraft position relative to the threshold altitude and is true if the aircraft position is below the threshold and false otherwise. (In this example, the threshold altitude is 2000 feet.) The other four monitored variables – mDOIstatus, mInitializing, mInhibit, and mReset – indicate whether the DOI is on or off, whether the system is being initialized, whether turning the DOI power on has been inhibited, and whether a system reset has been initiated.

As noted above, each mode in a mode class defines an equivalence class of system states. Modes are useful in defining the required relation REQ between the monitored and controlled variables. In the ASW, the system is in one of three modes: "init" if the system is initializing, "awaitDOIon" if the system is waiting for the DOI to power on, and "standby" otherwise.

Table 2. Monitored Variables of the ASW

Name	Type	Init. Value	Description
mAltBelow	boolean	true	below if alt. below threshold
mDOIStatus	enum	off	on if DOI powered on; else off
mInitializing	boolean	true	True iff system initializing
mInhibit	boolean	false	True iff DOI power on inhibited
mReset	boolean	false	True iff Reset button is pushed

Table 3 contains a mode transition table which specifies new values for the mode class "mcStatus" as a function of the five monitored variables listed in Table 2. The table states that the system transitions from "init" to "standby" when mInitializing becomes false and back to "init" when the user initiates Reset. It states further that the system transitions from "standby" to "awaitDOIon" if the aircraft altitude drops below the threshold altitude *when* the DOI is powered off *and* powering on the DOI is not inhibited. Finally, it states that the system transitions from "awaitDOIon" to "standby" when the DOI is powered on.

Table 3. Mode Transition Table for mcStatus

Mode Class mcStatus		
Old Mode	**Event**	**New Mode**
init	@F(mInitializing)	standby
standby	@T(mReset)	init
standby	@T(mAltBelow) WHEN NOT mInhibit AND mDOIStatus=off	awaitDOIon
awaitDOIon	@T(mDOIStatus=on)	standby
awaitDOIon	@T(mReset)	init

After the environmental variables and the mode class have been defined, the next step is to specify the required relation REQ. To do so, we define the value of the controlled variable cWakeupDOI as a function of the mode class mcStatus. As shown by Table 4, the value of cWakeupDOI depends solely on the current mode.

Table 4. Condition Table for cWakeupDOI

Mode Class mcStatus	
Mode	cWakeupDOI
init, standby	false
awaitDOIon	true

ASW System Design Specification

This section describes the input and output variables associated with the selected ASW input and output devices and the correspondence between these variables and the monitored and controlled variables specified above. To keep the paper concise, these input and output variables are described abstractly and details of the hardware device interfaces are omitted (e.g., whether the devices are interrupt-driven or polled, details of their control and data registers, etc.). However, these device details need to be recorded in the SDS eventually because the software developers need this information to design and write code for the device drivers.

Table 5. ASW Input and Output Variables

Name	Type	Init. Val.	Description
iAnaAltValue	yAltVal	below	below if alt. below threshold
iDigAlt1Value	yDigVal	0	a/c altitude measured in feet
iDigAlt2Value	yDigVal	0	a/c altitude measured in feet
iAnaAltStat	yAltStat	valid	Valid if altitude value reliable
iDigAlt1Stat	yAltStat	valid	Valid if altitude value reliable
iDigAlt2Stat	yAltStat	valid	Valid if altitude value reliable
iDOIStatus	enum	off	on if DOI powered on; else off
iInhibit	boolean	false	True iff DOI power on inhibited
iInitializing	boolean	true	True iff system initializing
iReset	boolean	false	True iff pilot has reset system
Time	integer	0	Time in milliseconds
oDOIPower	boolean	false	True iff DOI is to be powered on

Table 5 lists the 11 input variables and one output variable associated with the input and output devices of the ASW along with their types, initial values, and brief descriptions. Three altimeters, an analog altimeter and two digital altimeters, are used to estimate whether the aircraft is below the threshold altitude. The value provided by the analog altimeter, represented as "iAnaAltValue," indicates whether the aircraft is below or above the threshold. In contrast, the values provided by the two digital altimeters, represented as "iDigAlt1Value" and "iDigAlt2Value," specify the aircraft altitude in feet. The validity of the information provided by the altimeters is represented by the input variables "iAnaAltStat," "iDigAlt1Stat," and "iDigAlt2Stat". The integer input variable "Time" specifies the system time in milliseconds. The remaining input variables correspond to the remaining monitored variables, e.g., iDOIStatus corresponds to mDOIStatus, iInhibit corresponds to mInhibit, etc. The output variable oDOIPower is true if the device is to be powered on and false otherwise.

ASW Software Requirements Specification

As described above, the SoRS is organized into two device-dependent modules and a single device-independent module. Because the relation REQ in the SRS already defines the required behavior of the device-independent module, what remains is to specify the input and output device interface modules, i.e., D_IN and D_OUT.

The relation D_IN specifies how the input variables in Table 5 are used to compute estimates of the monitored variables in Table 2. (In our approach, estimates of the monitored and controlled variables are denoted as $\widetilde{\mathrm{mReset}}$, $\widetilde{\mathrm{mInhibit}}$, etc. To improve readability, we have omitted the tildes.) Estimating the values of four of the monitored variables – mDOIStatus, mInhibit, mReset, and mInitializing – from the corresponding input variables – iDOIStatus, iInhibit, iReset, and iInitializing – is straightforward. In each case, the estimated value of the monitored variable is simply the value of the corresponding input variable, i.e., mReset = iReset, mInhibit = iInhibit, etc. (These simple functions are placeholders for more complex functions for computing estimates of the monitored variables. Such information, e.g., how the software determines whether system initialization is complete, i.e., how and when mInitializing will be set to false, is unspecified in [1].)

Estimating the value of the monitored variable mAltBelow relies on the first six input variables listed in Table 5: the three values provided by the altimeters and the three indicators of validity of the altimeter information. If at least one of the altimeters indicates that the altitude is below the threshold and if that altimeter information is valid, then the ASW considers the altitude below the threshold, i.e., mAltBelow is true. Otherwise, mAltBelow is false. Table 6 contains a condition table that defines the value of mAltBelow as a function of the six altimeter inputs. In Table 6, kAlt represents the threshold altitude 2000 feet.

Relation D_OUT specifies how estimates of the values of the controlled variables are used to drive the output devices. In the ASW, the single controlled variable cWakeupDOI provides this estimate. The value of the output variable oDOIPower is simply a copy of the value of cWakeupDOI, i.e., oDOIPower = cWakeupDOI.

Table 6. Condition Table for mAltBelow

Condition	mAltBelow
(iAnaAltStat = valid AND iAnaAltValue = below) OR (iDigAlt1Stat = valid AND iDigAlt1Value <= kAlt) OR (iDigAlt2Stat=valid AND iDigAlt2Value <= kAlt)	true
(iAnaAltStat = invalid OR iAnaAltValue = above) AND (iDigAlt1Stat = invalid OR iDigAlt1Value > kAlt) AND (iDigAlt2Stat = invalid OR iDigAlt2Value > kAlt)	false

Reporting Hardware Malfunctions

This section adds behavior to the original SRS and to the SoRS to capture the reporting of hardware malfunctions. Three kinds of malfunctions are reported: malfunctions of the altimeters, malfunctions indicating initialization failure, and malfunctions in powering up the DOI. All three malfunctions are detected using two time-outs: FaultDur, which lasts 2 sec (or 2000 ms) and InitDur, which lasts 0.6 sec (or 600 ms). Adding the new behavior requires the creation of a new monitored variable, a new controlled variable, and a new output variable, and extensions to the mode transition table in Table 2 and the function defining cWakeupDOI in Table 4.

Table 7. Condition Table for mAltimeterFail

Condition	mAltimeterFail
iAnaAltStat = valid OR iDigAlt1Stat = valid OR iDigAlt2Stat = valid	false
iAnaAltStat = invalid AND iDigAlt1Stat = invalid AND iDigAlt2Stat = invalid	true

For each of the input variables, iAnaAltStat, iDigAlt1Stat, and iDigAlt2Stat, the value "invalid" indicates a malfunction in the corresponding altimeter. We define a new monitored variable "mAltimeterFail" to represent the malfunction of all three altimeters. Table 7 contains a condition table defining mAltimeterFail as a function of these three input variables.

Table 8 extends the mode transition table in Table 3 to capture the detection of hardware malfunctions. To represent these malfunctions, a new mode called "fault" is added to the set of modes in the mode class mcStatus. Table 8 states that a fault is detected if the system remains in the initializing state for more than 0.6 sec, if the DOI takes more than 2 sec to power up, or if all three altimeters have failed for 2 sec. The system recovers from malfunctions when the pilot presses Reset. To mark the extensions to Table 2, we have shaded the four transitions in Table 8 which involve the mode "fault" and the added constraint for system transfer from "standby" to "awaitDOIon".

Table 8. Extended Mode Transition Table

Mode Class mcStatus		
Old Mode	**Event**	**New Mode**
init	@F(mInitializing)	standby
init	@T(DUR(mcStatus=init) > InitDur)	fault
standby	@T(mReset)	init
standby	@T(mAlt_Below) WHEN NOT mInhibit AND mDOIStatus=off AND NOT mAltimeterFail	awaitDOIon
standby	@T(DUR(mAltimeterFail) > FaultDur)	fault
awaitDOIon	@T(mDOIStatus=on)	standby
awaitDOIon	@T(mReset)	init
awaitDOIon	@T(DUR(mcStatus = awaitDOIon) > FaultDur) OR @T(DUR(mAltimeterFail) > FaultDur)	fault
fault	@T(mReset)	init

The extension of the ASW to report hardware malfunctions requires modifications to Table 4, which defines the value of cWakeupDOI, and the creation of two new variables, a new controlled variable to turn on a fault indicator light and a new output variable to strobe the watchdog timer. Table 9 is a trivial extension of Table 4 to cover the new mode "fault". Table 10 contains a function defining the value of a new controlled variable called cFaultIndicator which is on if the system is in the mode "fault" and off otherwise.

Table 9. New Condition Table for cWakeupDOI

Mode of mcStatus	cWakeupDOI
init, standby, fault	false
awaitDOIon	true

Table 10. Condition Table for cFaultIndicator

Mode of mcStatus	cFaultIndicator
init, standby, awaitDOIon	off
fault	on

Table 11 defines a function to strobe the watchdog timer when no fault has been detected. The first row states that oWatchDogTimer is set to true when it has remained false for 0.1 sec and cFaultIndicator is false. The second row describes a similar transition setting the variable to false. This ensures that the system strobes the watchdog timer every 0.2 sec when no fault has been detected.

Table 11. Event Table for oWatchDogTimer

Event	oWatchDogTimer’
@T(DUR(NOT oWatchDogTimer) >= 100) WHEN NOT cFaultIndicator	true
@T(DUR(oWatchDogTimer) >= 100) WHEN NOT cFaultIndicator	false

Applying the SCR Tools

We used the SCR* toolset to develop the SCR specification of the required ASW behavior and to analyze the specification for desired properties. The analysis tools that we applied were the consistency checker and the simulator. We performed the analysis in two stages. First, we used the tools to analyze the SRS, i.e., the ideal system behavior. Once we had confidence in the quality of this specification, we added the refinements and extensions described above -- the input and output variables and the extensions to the specification to report hardware malfunctions.

Our analysis uncovered some minor errors, which we corrected with only a small investment in human effort and time. Further, running the consistency checker and finding no problems, and running a series of scenarios through the simulator and finding that the simulated behavior matched our expectations, increased our confidence in the specification’s correctness. Moreover, our analysis exposed an error in the specification in [1] and raised important questions. For example, what happens if an inhibit signal is received when the ASW is in mode “awaitDOIon?” Should there be more than a single fault indicator light, perhaps one for each altimeter?

Summary

In our view, separation of the ideal system behavior from the real system behavior (i.e., the behavior that includes data associated with I/O devices and hardware malfunctions) is very valuable in requirements specification and analysis. The specification of the ideal behavior of the ASW is very simple: it involves only seven variables and four tables and is presented on a single page. In our view, understanding and specifying the essential system behavior (captured by the SRS) should precede the consideration of details such as the characteristics of input and output devices and the reporting of hardware malfunctions. Adding the extra information to the SDS and the SoRS and extending the system behavior is straightforward once the essential behavior is understood.

Alternative specifications of the required ASW behavior are presented by Miller in [1] and by Thompson et al. in [7]. Miller does not specify the ideal system behavior and hence we found his specification difficult to understand and to analyze. The specification in [7] also combines the ideal system behavior with the externally visible behavior which reports hardware malfunctions but, like our approach, uses refinement and extension to produce the complete specification. For another example in which we applied our four-step process to a more complex system specification, see [8].

Acknowledgement: We thank Beth Leonard for her valuable comments on drafts of this paper.

References

[1] S. Miller. Modeling software requirements for embedded systems. Draft report, 1999.

[2] C. Heitmeyer et al. SCR*: A toolset for specifying and analyzing software requirements. *In Proc. 10th Computer-Aided Verification Conf.,* Vancouver, Canada, 1998.

[3] C. Heitmeyer et al. Automated consistency checking of requirements specifications. *ACM Trans. On Software Eng. and Methodology* (5)3: 231-261, 1996.

[4] S. Easterbrook et al. Formal methods for verification and validation of partial specifications: A case study. *Journal of Systems and Software,* 1997.

[5] S. Miller. Specifying the mode logic of a flight guidance system. In *Proc. 2nd ACM Workshop on Formal Methods in Software Practice* (FMSP ’98). 1998.

[6] D. Parnas et al. Functional documentation for computer systems. *Science of Computer Programming 25(1).* 1995.

[7] J. Thompson et al. Specification-based prototyping for embedded systems. In *Proc. 7th ESEC/FSE,* Sept. 1999.

[8] C. Heitmeyer and R. Bharadwaj. Applying the SCR requirements method to the Light Control case study. *Journal of Universal Computer Science.* Aug. 2000 (to appear).